Hello in this tutorial, we will create a file upload functionality through python flask and host it on Docker in a containerized environment.

<h2>1. Introduction</h2>

A common feature of any web application is to provide support to the file upload utility. In python programming, flask provides this functionality where all we need is an HTML form with the encryption level as <code>multipart/form-data</code>. The server-side script will fetch the file from the <code>requests.files[]</code> object and on successful uploading, it will be saved in the desired folder on the server.

<h3>1.1 Setting up Python</h3>

If someone needs to go through the Python installation on Windows, please watch <a href="https://www.youtube.com/watch?v=i-MuSAwgwCU" target="\_blank">this</a> link. You can download the Python from <a href="https://www.python.org/downloads/" target="\_blank">this</a> link.

<h3>1.2 What is Docker & Setting up Docker</h3>

In the present world, <strong>Docker</strong> is an important term,

<ul>

<li>Often used in CI/CD platform that packages and runs the application with its dependencies inside a container</li>

<li>Is a standard for Linux Containers</li>

<li>A <em>Container</em> is a runtime that runs under any Linux kernel and provides a private machine-like space under Linux</li>

</ul>

If someone needs to go through the Docker installation, please watch <a href="https://www.youtube.com/watch?v=R-ZMkGvh-9Y" target="\_blank">this</a> video.

<h2>2. Upload a File with Python Flask</h2>

I am using <a href="https://www.jetbrains.com/pycharm/" target="\_blank">JetBrains PyCharm</a> as my preferred IDE. You're free to choose the IDE of your choice. Fig. 1 represents the project structure for this tutorial.

// Fig. 1: Application structure

<h3>2.1 Creating a requirements file</h3>

Add the below code to the requirements file. The file will be responsible to download and install the packages required for this tutorial.

<span style="text-decoration: underline;"><em>requirements.txt</em></span>

<pre class="brush:plain;">

Flask  
Werkzeug

</pre>

<h3>2.2 Basic file upload from</h3>

Define an HTML form in the <code>templates</code> folder with a <code>file</code> field in it. The page will also show the success or error messages from the python flask backend.

<span style="text-decoration: underline;"><em>upload.html</em></span>

<pre class="brush:plain;">

&lt;!DOCTYPE html&gt;

&lt;html lang="en"&gt;

&lt;head&gt;

&lt;meta charset="UTF-8" /&gt;

&lt;title&gt;File upload&lt;/title&gt;

&lt;/head&gt;

&lt;body&gt;

&lt;h1&gt;Python flask file upload&lt;/h1&gt;

&lt;hr&gt;

&lt;p&gt;

{% with messages = get\_flashed\_messages() %}

{% if messages %}

&lt;ul class=flashes&gt;

{% for message in messages %}

&lt;li&gt;{{ message }}&lt;/li&gt;

{% endfor %}

&lt;/ul&gt;

{% endif %}

{% endwith %}

&lt;/p&gt;

&lt;h3&gt;Select a file to upload&lt;/h3&gt;

&lt;form action="/" method="POST" enctype="multipart/form-data"&gt;

&lt;input type="file" name="file" /&gt;

&lt;input type="submit" value="Submit" /&gt;

&lt;/form&gt;

&lt;/body&gt;

&lt;/html&gt;

</pre>

<h3>2.3 Python code</h3>

The python file will expose three different endpoints responsible for interacting with the frontend page. We will also add some basic validations such as max file size and allowed file type to the code which can be referenced in the same file.

<ul>

<li><code>/</code> endpoint - Responsible for showing the HTML page to the user via the HTTP GET call. The same endpoint will also be responsible to handle the HTTP POST call from the form and save the uploaded file in the <code>uploads</code> folder</li>

<li><code>/download</code> endpoint - Responsible to handle the download functionality. I have skipped the download functionality for users to play with the application</li>

</ul>

<span style="text-decoration: underline;"><em>main.py</em></span>

<pre class="brush:python;">

import logging

import os.path

from flask import Flask, render\_template, request, redirect, flash

from werkzeug.utils import secure\_filename

# [logging config

logging.basicConfig(format='%(asctime)s:%(levelname)s:%(filename)s:%(funcName)s:%(message)s',

datefmt='%Y-%m-%d %H:%M:%S',

level=logging.INFO)

# logging config]

app = Flask(\_\_name\_\_)

app.secret\_key = "somesecretkey"

app.config['ALLOWED\_EXTENSIONS'] = ['.jpg', '.png']

app.config['MAX\_CONTENT\_LENGTH'] = 1024 \* 1024

UPLOAD\_FOLDER = os.path.join(os.getcwd(), 'uploads')

@app.route('/', methods=['GET'])

def index():

logging.info('Showing index page')

return render\_template('upload.html')

@app.route('/', methods=['POST'])

def upload\_files():

logging.info('Starting file upload')

if 'file' not in request.files:

flash('No file part')

return redirect(request.url)

file = request.files['file']

# obtaining the name of the destination file

filename = file.filename

if filename == '':

logging.info('Invalid file')

flash('No file selected for uploading')

return redirect(request.url)

else:

logging.info('Selected file is= [%s]', filename)

file\_ext = os.path.splitext(filename)[1]

if file\_ext in app.config['ALLOWED\_EXTENSIONS']:

secure\_fname = secure\_filename(filename)

logging.info('Secure filename is= [%s]', secure\_fname)

file.save(os.path.join(UPLOAD\_FOLDER, secure\_fname))

logging.info('Upload is successful')

flash('File uploaded successfully')

return redirect('/')

else:

logging.info('Invalid file extension')

flash('Not allowed file type')

return redirect(request.url)

# Todo - To be implemented.

@app.route('/download', methods=['GET'])

def download():

return 'Download file'

def check\_upload\_dir():

if not os.path.exists(UPLOAD\_FOLDER):

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

if \_\_name\_\_ == '\_\_main\_\_':

check\_upload\_dir()

# Development only: run "python app.py" and open http://localhost:5000

server\_port = os.environ.get('PORT', '5000')

app.run(debug=False, port=server\_port, host='0.0.0.0')

</pre>

For localhost debugging, you can run the application by running the <code>main.py</code> py file in the PyCharm IDE. The application will be started on the <code>5000</code> port number and you can hit the following url - <code>http://localhost:5000</code> in the browser of your choice to display the HTML page.

<h2>3. Setting up requirements for Docker</h2>

To deploy this application on Docker we will create two files responsible for having this application up and running in a container. We will also use the <code>.dockerignore</code> file so that we only add the required files to the docker image and the same file can be downloaded from the Downloads section.

<h3>3.1 Creating a Dockerfile</h3>

Add the below code to the Dockerfile responsible for creating the Docker image. The image name will be driven from the <code>docker-compose.yml</code> file which will be created in the next step.

<span style="text-decoration: underline;"><em>Dockerfile</em></span>

<pre class="brush:plain;">

# Python image to use

FROM python:3.10-slim

# Author of the image

MAINTAINER danielatlas

# Set the working directory to /app

WORKDIR /app

# Copy the requirements file used for dependencies

COPY requirements.txt .

# Install the needed packages specified in requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

# Copy rest of the working directory contents into the container at /app

COPY ../upload .

# Run app.py when the container launches

ENTRYPOINT ["python3", "main.py"]

</pre>

<h3>3.2 Creating the docker-compose yml</h3>

Add the below code to the docker-compose file responsible for bundling the code, creating the docker image, and starting the container on a port number - <code>5000</code> from the created image. You are free to change the details as per your wish but remember that if you change the port number hit the application endpoint on the same port. For this tutorial, I am setting the port number to <code>5000</code>.

<span style="text-decoration: underline;"><em>docker-compose.yml</em></span>

<pre class="brush:plain;">

version: "3.7"

services:

pyflaskfileupload:

build:

context: ../upload

dockerfile: Dockerfile

image: "pyflaskfileupload:latest"

container\_name: "pyflaskfileupload"

ports:

- "5000:5000"

</pre>

<h2>4. Code run & demo</h2>

To run the application on the docker container follow the below steps. Open the terminal and navigate to the project directory containing the Dockerfile.

<span style="text-decoration: underline;"><em>App run</em></span>

<pre class="brush:plain;">

-- to create the image and start container --

docker-compose up -d --build

-- to check if the container is started

docker ps -a

-- to stop and remove the container --

docker-compose down

-- to view the container logs --

docker logs --follow pyflaskfileupload

-- to remove the created image --

docker rmi pyflaskfileupload

</pre>

Once the <em>up</em> command is successful you can hit the following endpoint - <code>http://localhost:5000</code> in the browser of your choice and the below page will appear.

// Fig. 2: Application demo

An HTML form like in Fig. 2 will be shown to the users and the user can browse the file system for the file and will be uploaded to the server inside the <code>uploads</code> folder. Once the file is uploaded successfully it will be shown in the directory.

// Fig. 3: Uploaded file

That is all for this tutorial and I hope the article served you with whatever you were looking for. Happy Learning and do not forget to share!

<h2>5. Summary</h2>

In this tutorial, we learned about the file upload functionality in a python application using the flask module and hosted the same app on docker. You can download the source code of this tutorial from the <a href="#projectDownload">Downloads</a> section.

<h2><a name="projectDownload"></a>6. Download the Project</h2>

This was a tutorial on how to create a python flask file upload functionality and run it in the localhost and dockerized environment.

<div class="download"><strong>Download</strong><br />You can download the full source code of this example here: </div>

=============================================================================================

Application name - <strong>Upload a File with Python Flask</strong>